LECTURE 6

CONDITIONALS & BOOLEAN ALGEBRA

MCS 260 Fall 2020

Emily Dumas

REMINDERS

- No class Monday (Labor day)
- Quiz 2 due 6pm central on Tue Sep 8
- Project 1 description will be posted today
    - Project 1 due Fri Sep 18

A TRICKY LIST

What do you expect this code to do?

Will it produce an error?

```
L = ["a","b","c"]
L[0] = L
print(L)
```

CONDITIONALS

You can indicate that a section of code should only execute if certain conditions are met.

Syntax:

```
if condition:
    statement
    statement
    ...
statement that runs regardless of the condition
```

Indenting statements below **`if`** by the same amount makes them a **code block**. The block ends when a line is vertically aligned with **`if`**.

In many other languages, special symbols are used to indicate the start and end of a block, and indenting is ignored.

`{` and `}` are common choices.

Python uses indenting as a substitute for block start / block end symbols.

```python
n = int(input("How many penguins live with you? "))
if n > 150:
    print("That's quite a crowd!")
print("Thank you for completing the penguin census.")
```

This example uses four spaces to indent. That is the recommended (and most popular) number.

SPACES VS TABS

The code point `U+0009` is "CHARACTER TABULATION", better known as "tab".

Python *allows* this character to be used for indenting, but recommends against it, and *forbids* mixing spaces and tabs.

Depending on your editor, pressing the Tab key may:

- Insert a fixed number of spaces
- Insert a context-dependent number of spaces
- Insert `U+0009`

Recommendation for Python coding:

Configure your editor to never insert `U+0009`*.*

This is often the default behavior.

CONDITIONS

Python supports a lot of conditions (tests) that can appear in an `if` statement, e.g. comparison operators:

`>`	is greater than
`<`	is less than
`==`	is equal to *note two equal signs!*
`!=`	is not equal to

`>=`	is greater than or equal to
`<=`	is less than or equal to

ELSE

An `if` statement can be followed by `else:` and a code block to be executed if the condition is False.

```
if x == 100:
    print("x is equal to 100")
else:
    print("x is NOT equal to 100")
```

This is useful for handling dichotomies.

ELIF

An **`if`** statement can also be followed by **`elif`** (for "else if"), which begins a new conditional.

```
if x == 100:
    print("x is equal to 100")
elif x % 4 == 0:
    print("x is a multiple of 4, but is not equal to 100")
elif x % 2 == 0:
    print("x is even, but is not a multiple of 4")
else:
    print("x is odd")
```

A chain of **`if`**/**`elif`**/**`elif`**/`...` is the typical way to compare a variable to multiple values or categories.

Example: quadroots.py

```python
# Determine the number of real roots of a quadratic polynomial
# MCS 260 Fall 2020 Lecture 6 - Emily Dumas
print("Enter the coefficients a,b,c of ax^2+bx+c, one per line.")
a = float(input())
b = float(input())
c = float(input())

print("You entered:",a,"x^2 +",b,"x +",c)

discriminant = b**2 - 4*a*c

if discriminant > 0:
    print("This polynomial has two real roots.")
elif discriminant == 0:
    print("This polynomial has exactly one real root.")
else:
    # Now we know discriminant < 0
    print("This polynomial doesn't have any real roots.")
```

MORE CONDITIONS

x `in` seq	Sequence seq contains an item equal to x
x `not in` seq	(negation of above)
$cond_0$ `and` $cond_1$	Both $cond_0$ and $cond_1$ are True.
$cond_0$ `or` $cond_1$	At least one of $cond_0$ and $cond_1$ is True.
`not` $cond$	$cond$ is False.

PRECEDENCE

Comparison operators all have lower precedence than arithmetic, so e.g. `5*5>30-10` evaluates as True. The order is:

1. Arithmetic (PEMDAS)
2. `>, >=, <, <=`
3. `==, !=`
4. **`in, not in`**
5. **`and, or, not`**

BOOL

`bool`, for "boolean", is a type that has only two possible values, **`True`** and **`False`**.

Conditions in **`if`** or **`elif`** actually evaluate as **`bool`**s, and you can have **`bool`** variables, too.

```python
everything_will_be_ok = True
missed_quiz_deadline = False
x = 1 < 2    # x is now True
y = 3 > 4    # y is now False
if x and not y:
    print("Good news: math is not broken.")
```

BOOLEAN ALGEBRA

Booleans are also considered in math / theoretical CS.

Different symbols are often used for boolean operators:

$x \land y$	means	x and y
$x \lor y$	means	x or y
$\neg x$	means	not x

In addition, $\bar{x}$ or $!x$ are sometimes used for $\neg x$.

The operators $\wedge$ and $\vee$ are commutative and associative. They obey algebraic rules such as:

- $\neg(\neg x) = x$
- $x \vee x = x$ and $x \wedge x = x$
- $x \vee (\neg x) = \textbf{\texttt{True}}, x \wedge (\neg x) = \textbf{\texttt{False}}$
- $x \vee \textbf{\texttt{True}} = \textbf{\texttt{True}}, x \vee \textbf{\texttt{False}} = x,$ $x \wedge \textbf{\texttt{False}} = \textbf{\texttt{False}}, x \wedge \textbf{\texttt{True}} = x.$
- Distributive law: $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
- DeMorgan's law: $\neg(x \wedge y) = (\neg x) \vee (\neg y),$ $\neg(x \vee y) = (\neg x) \wedge (\neg y)$

Once you decode what these rules are saying, all but the named ones will probably become obvious.

If I ever ask you to perform boolean algebra simplification, I will provide this list.

These rules can be used to simplify boolean expressions, e.g.

$x\ \mathtt{and\ not}\ (x\ \mathtt{and}\ y)$	
$\rightarrow\ x \wedge \neg(x \wedge y)$	Math notation
$\rightarrow\ x \wedge ((\neg x) \vee (\neg y))$	DeMorgan
$\rightarrow\ (x \wedge (\neg x)) \vee (x \wedge (\neg y))$	Distributive
$\rightarrow\ \mathtt{False} \vee (x \wedge (\neg y))$	
$\rightarrow\ x \wedge (\neg y)$	
$\rightarrow\ x\ \mathtt{and\ not}\ y$	

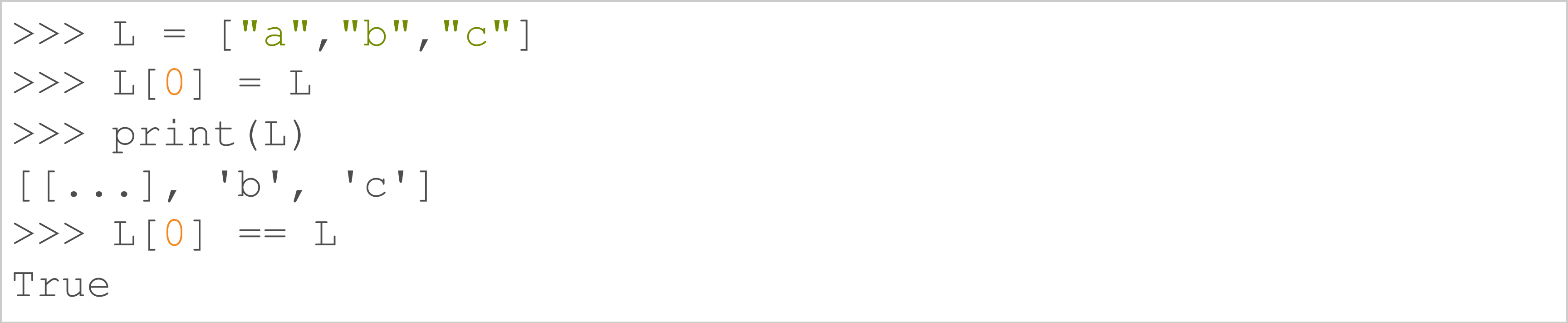
BACK TO THE TRICKY LIST

What do you expect this code to do?

Will it produce an error?

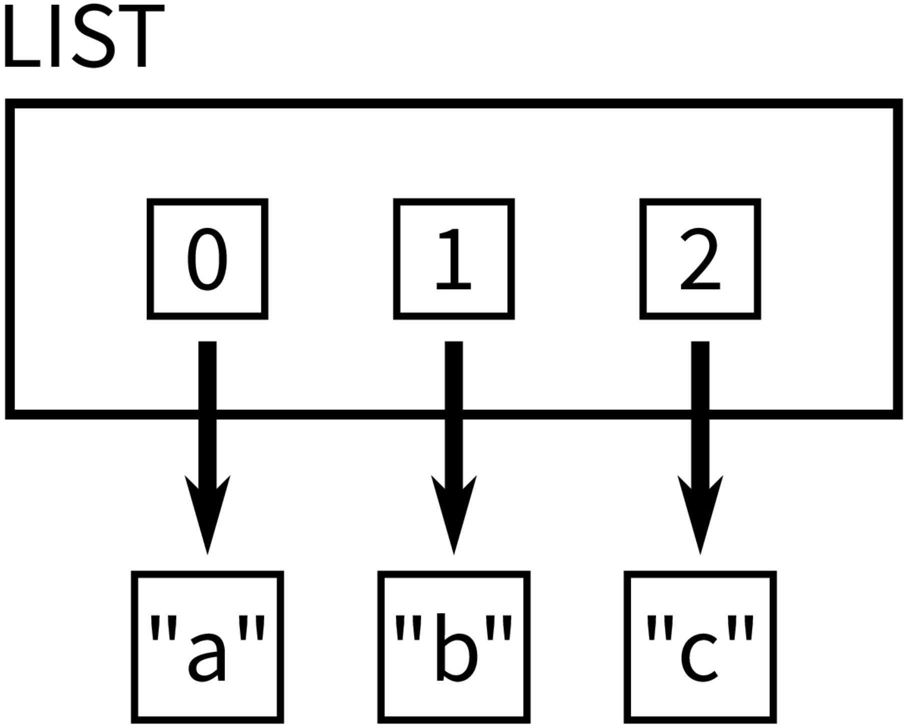
```
L = ["a","b","c"]
L[0] = L
print(L)
```

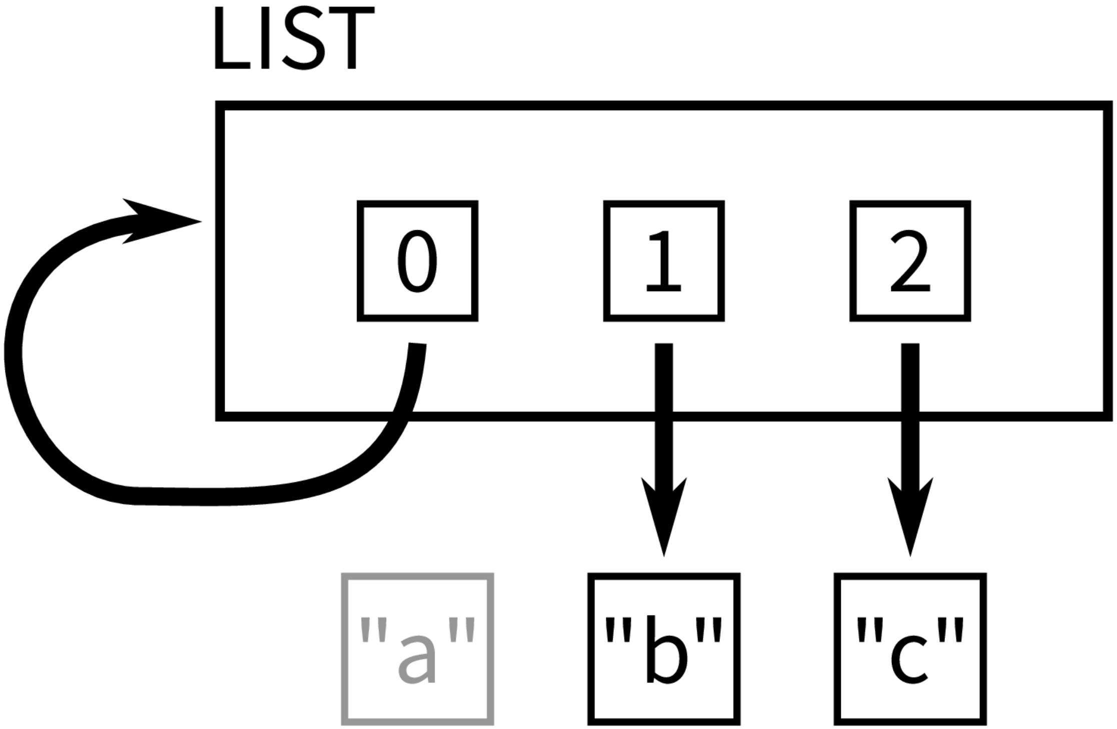
Answer: No error. A list in Python can contain itself.



```
>>> L = ["a","b","c"]
>>> L[0] = L
>>> print(L)
[[...], 'b', 'c']
>>> L[0] == L
True
```

The "..." is there so that the print function doesn't get stuck constructing an infinite output!


LIST
0
1
2
"a"
"b"
"c"


LIST
0
1
2
"a"
"b"
"c"

REFERENCES

- In *Downey*:
  - Conditionals and booleans are discussed in sections 5.1 - 5.7.

REVISION HISTORY

- 2020-09-04 Typo fix
- 2020-09-04 Initial publication